勇闯迷宫游戏

作 者 姓 名： 李翠琪

学 号： 1751022

指 导 教 师： 张颖 \_\_

学院、 专业： 软件工程 \_\_

同济大学

Tongji University

目录

[1 分析 1](#_Toc24372736)

[1.1 项目简介 1](#_Toc24372737)

[1.2 功能要求 1](#_Toc24372738)

[2 设计 2](#_Toc24372739)

[2.1 Node设计 2](#_Toc24372740)

[2.2 Stack设计 2](#_Toc24372741)

[2.3 主程序设计 2](#_Toc24372742)

[3 实现 2](#_Toc24372743)

[3.1 Node类实现 2](#_Toc24372744)

[3.2 Stack类实现 3](#_Toc24372745)

[3.2.1 push功能实现 3](#_Toc24372746)

[3.2.2 pop功能实现 3](#_Toc24372747)

[3.2.3 empty，top功能实现 3](#_Toc24372748)

[3.3 主函数功能的实现 4](#_Toc24372749)

[3.3.1 输入的处理 4](#_Toc24372750)

[3.3.2 isValid（）函数 4](#_Toc24372751)

[3.3.3 dfs()函数初步实现 5](#_Toc24372752)

[3.3.4 dfs完整实现 6](#_Toc24372753)

[4 测试 8](#_Toc24372754)

[4.1 功能测试 8](#_Toc24372755)

[4.1.1测试1 8](#_Toc24372756)

[4.1.2 测试2 9](#_Toc24372757)

[4.1.3 测试3 10](#_Toc24372758)

[4.2 边界测试 11](#_Toc24372759)

[4.2.1 没有从入口到出口的路径 11](#_Toc24372760)

[4.2.2 入口不在数组中或者为墙 12](#_Toc24372761)

[4.2.3 出口不在迷宫中或者出口为墙 14](#_Toc24372762)

# 1 分析

## 1.1 项目简介

迷宫只有两个门，一个门叫入口，另一个门叫出口。一个骑士骑马从入口进入迷宫，迷宫设置很多障碍，骑士需要在迷宫中寻找通路以到达出口。

## 1.2 功能要求

迷宫问题的求解过程可以采用回溯法即在一定的约束条件下试探地搜索前进，若前进中受阻，则及时回头纠正错误另择通路继续搜索的方法。从入口出发，按某一方向向前探索，若能走通，即某处可达，则到达新点，否则探索下一个方向；若所有的方向均没有通路，则沿原路返回前一点，换下一个方向再继续试探，直到所有可能的道路都探索到，或找到一条通路，或无路可走又返回入口点。在求解过程中，为了保证在达到某一个点后不能向前继续行走时，能正确返回前一个以便从下一个方向向前试探，则需要在试探过程中保存所能够达到的每个点的下标以及该点前进的方向，当找到出口时试探过程就结束了。

实际就是一个深度优先搜索,只不过不是在图中,而是在数组中,所有只要注意数组边界就好。同时，需要存储迷宫的路径，所以还需要一个栈。

# 2 设计

## 2.1 Node设计

Node节点和之前的类似,它保存一个模板类型的元素value,并且有一个指针next指向下一个元素,都是public的,以便于访问。

## 2.2 Stack设计

底层存储Node类头节点以及栈的元素个数N。为外界提供公有函数push（），pop（），top（），empty（）。提供栈的基本功能。

## 2.3 主程序设计

由用户输入迷宫的规模以及迷宫具体地形，再让用户输入起始位置及目标位置。并且程序判断位置是否合理。展示迷宫原来的样子，再通过dfs（）递归寻找出路，并且用一个栈存储路径，如果找到了从栈中取出路径（颠倒一次栈），如果没找到给出没有路径的提示。

# 3 实现

## 3.1 Node类实现

template <class T> class Node  
{  
public:  
  
 Node(const T & value, Node<T> \*next= nullptr) : value(value), next(next) {}  
 Node(Node<T> \*next= nullptr) : next(next) {}  
  
 T value;  
 Node<T> \*next;  
};

## 3.2 Stack类实现

### 3.2.1 push功能实现

新加入一个Node元素，并且将之指向原来的头节点，原来的头节点指向现在的头节点，将存储的元素个数N+1。

void push(T t)  
 {  
 Node<T> \*oldFirst = first;  
 first = new Node<T>(t, oldFirst);  
++N;  
 }

### 3.2.2 pop功能实现

判断是否为空，不为空就让头节点指向下一个元素，然后delete之前的元素，释放内存，--N

void pop()  
 {  
 if(empty())  
 return;  
 Node<T> \*temp = first;  
 first = first->next;  
 delete temp;  
--N;  
 }

### 3.2.3 empty，top功能实现

bool empty()  
{  
 return N==0;  
}  
  
T top()  
{  
 return first->value;  
}

## 3.3 主函数功能的实现

### 3.3.1 输入的处理

先根据输入,赋值给M,N,然后创建M\*N大小的二维数组,再根据输入的内容创建迷宫地形。

int M,N;  
cout << "先输入两个正整数,代表迷宫大小M\*N;再输入M\*N正整数(0,1代表迷宫的路和墙):";  
cin >> M >> N;  
 vector<vector<int>> map(M, vector<int>(N,1));//默认设置全为墙,当然没什么用,因为是用户进行输入  
 vector<vector<bool>> marked(M, vector<bool>(N, false));//默认为没访问过的

/\*  
 \* 0 0 路  
 \* 1 # 墙  
 \* 2 x 走的路径  
 \*/  
for (int i = 0; i < M; i++)  
{  
 for (int j = 0; j < N; j++)  
 {  
 int mapType;  
 cin >> mapType;  
 map[i][j] = mapType;  
 }  
}

然后输入起始，终止位置

cout << "输入起始坐标和结束坐标(xlo, ylo), (xhi, yhi):";  
int xlo, ylo, xhi, yhi;  
cin >> xlo >> ylo >> xhi >> yhi;

### 3.3.2 isValid（）函数

用于检查起始，终止位置合理性，是否越界，是否为墙，并且给相应的提示，如果不合法，就直接结束程序。

bool isValid(vector<vector<int>> & map,int M,int N,int xlo,int ylo,int xhi,int yhi)  
{  
 if (xlo < 0 || xlo >= M || ylo < 0 || ylo >= N)  
 {  
 cout << "起始路径就不在迷宫里面啊,大哥";  
 }  
 else if (map[xlo][ylo] == 1)//输入为墙,搞什么  
 {  
 cout << "起始路径为墙,你让他怎么走?";  
 }  
 else if (xhi < 0 || xhi >= M || yhi < 0 || yhi >= N)  
 {  
 cout << "终点就不在迷宫里面啊,大哥";  
 }  
 else if (map[xhi][yhi] == 1)//终点为墙,搞什么  
 {  
 cout << "终点为墙,你让他怎么到?";  
 }  
 else  
 {  
 return true;  
 }  
 return false;  
}

### 3.3.3 dfs()函数初步实现

从xlo,ylo开始，在没有找到xhi,yhi之前，不懂调用递归往四个方向的，没有标记为访问过的，没有越界的，是路径的位置走，而且是深度优先搜索，即如果进入了一个方向，则直到找完这个方向的所有路径后，才走另一个方向。

初步实现的关键地方的代码（因为是直接截取完整程序，所以参数会带上了之后所用到的栈）：

marked[i][j] = true;

if(i-1>=0 && map[i-1][j]==0 && !marked[i-1][j])  
{  
 dfs(map, marked,path, i - 1, j, xhi, yhi, hasPath);  
}  
if(i+1<M && map[i+1][j]==0 && !marked[i+1][j])  
{  
 dfs(map, marked,path, i+1, j, xhi, yhi, hasPath);  
}  
if(j+1<N && map[i][j+1]==0 && !marked[i][j+1])  
{  
 dfs(map, marked,path, i, j+1, xhi, yhi, hasPath);  
}  
if(j-1>=0 && map[i][j-1]==0 && !marked[i][j-1])  
{  
 dfs(map, marked,path, i, j-1, xhi, yhi, hasPath);  
}

### 3.3.4 dfs完整实现

初步实现后，为了增加显示迷宫路径功能以及没有路径时给出提示的功能，还用到了一个自己实现的栈，以及bool类型的hasPath,用于当没有路径时才输出没有路径的提示信息。

每递归调用一次，就将这个坐标加入栈中（用pair存储了坐标）。而当四个if都没进入时，即无路可走时，就弹出栈顶元素，这意味着，这个栈总是存储着当前能走的路径。

path.push(make\_pair(i, j));

//四个if，不写出来了

//无路可走时,弹栈  
path.pop();

然后，如果递归到了终点位置，即有路可走，就将hasPath设为true，并且用一个栈存储当前栈的反转（这样才是从入口到出口）。并且还将map在这个路径上的元素设置为2，即为路径。

最后，输出迷宫，迷宫上的路径用（X）表示，并且在下方通过遍历那个反转栈，输出路径坐标，从入口到出口。结束dfs（）函数。

void dfs(vector<vector<int>> & map, vector<vector<bool>>& marked,Stack<pair<int,int>>& path, int i, int j, int xhi,int yhi,bool & hasPath)  
{  
 int M = map.size(), N = map[0].size();  
 path.push(make\_pair(i, j));  
 marked[i][j] = true;  
 if(i==xhi && j==yhi)  
 {  
 hasPath = true;  
 Stack<pair<int, int>> reverse;  
 while(!path.empty())  
 {  
 pair<int, int> p = path.top();  
 path.pop();  
 reverse.push(p);//反转的stack,用于输出路径  
 map[p.first][p.second] = 2;  
 }  
  
 cout<<endl;  
 cout << " ";  
 for (int i = 0; i < N; ++i)  
 {  
 cout << i << "列 ";  
 }  
 cout<<endl;  
 for (int i = 0; i < M; ++i)  
 {  
 cout << i << "行 ";  
 for (int j = 0; j < N; ++j)  
 {  
 if(map[i][j]==0)  
 {  
 cout << "0";  
 }  
 else if(map[i][j]==2)  
 {  
 cout << "X";  
 }  
 else  
 {  
 cout << "#";  
 }  
 cout << " ";  
 }  
 cout<<endl;  
 }  
  
 cout<<endl<<"迷宫路径为:"<<endl;  
 while(!reverse.empty())  
 {  
 pair<int, int> p = reverse.top();  
 reverse.pop();  
 if(p.first==xhi && p.second==yhi)  
 cout<<"(" << p.first << "," << p.second<<")"<<endl;  
 else  
 cout << "(" << p.first << "," << p.second << ")->";  
 }  
 }  
 if(i-1>=0 && map[i-1][j]==0 && !marked[i-1][j])  
 {  
 dfs(map, marked,path, i - 1, j, xhi, yhi, hasPath);  
 }  
 if(i+1<M && map[i+1][j]==0 && !marked[i+1][j])  
 {  
 dfs(map, marked,path, i+1, j, xhi, yhi, hasPath);  
 }  
 if(j+1<N && map[i][j+1]==0 && !marked[i][j+1])  
 {  
 dfs(map, marked,path, i, j+1, xhi, yhi, hasPath);  
 }  
 if(j-1>=0 && map[i][j-1]==0 && !marked[i][j-1])  
 {  
 dfs(map, marked,path, i, j-1, xhi, yhi, hasPath);  
 }  
 //无路可走时,弹栈  
 path.pop();  
}

# 4 测试

## 4.1 功能测试

### 4.1.1测试1

**测试用例**：

7 7

0 0 1 1 1 1 1

1 0 0 0 1 0 1

0 0 1 0 1 1 1

1 0 0 1 1 1 1

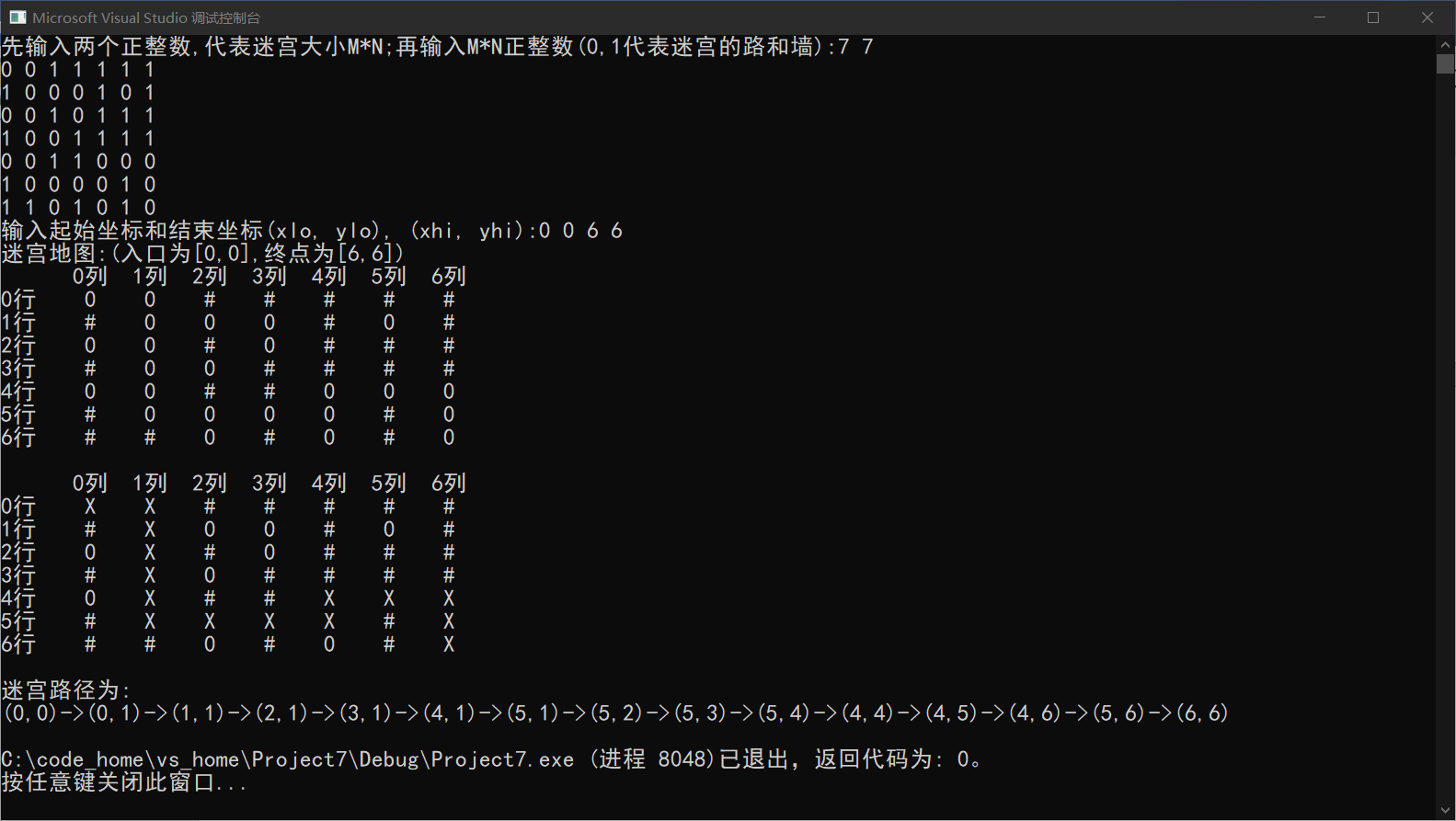
0 0 1 1 0 0 0

1 0 0 0 0 1 0

1 1 0 1 0 1 0

0 0 6 6

**实验结果**



### 4.1.2 测试2

**测试用例：**

7 7

0 0 1 1 1 1 1

1 0 0 0 1 0 1

0 0 1 0 1 1 1

1 0 0 1 1 1 1

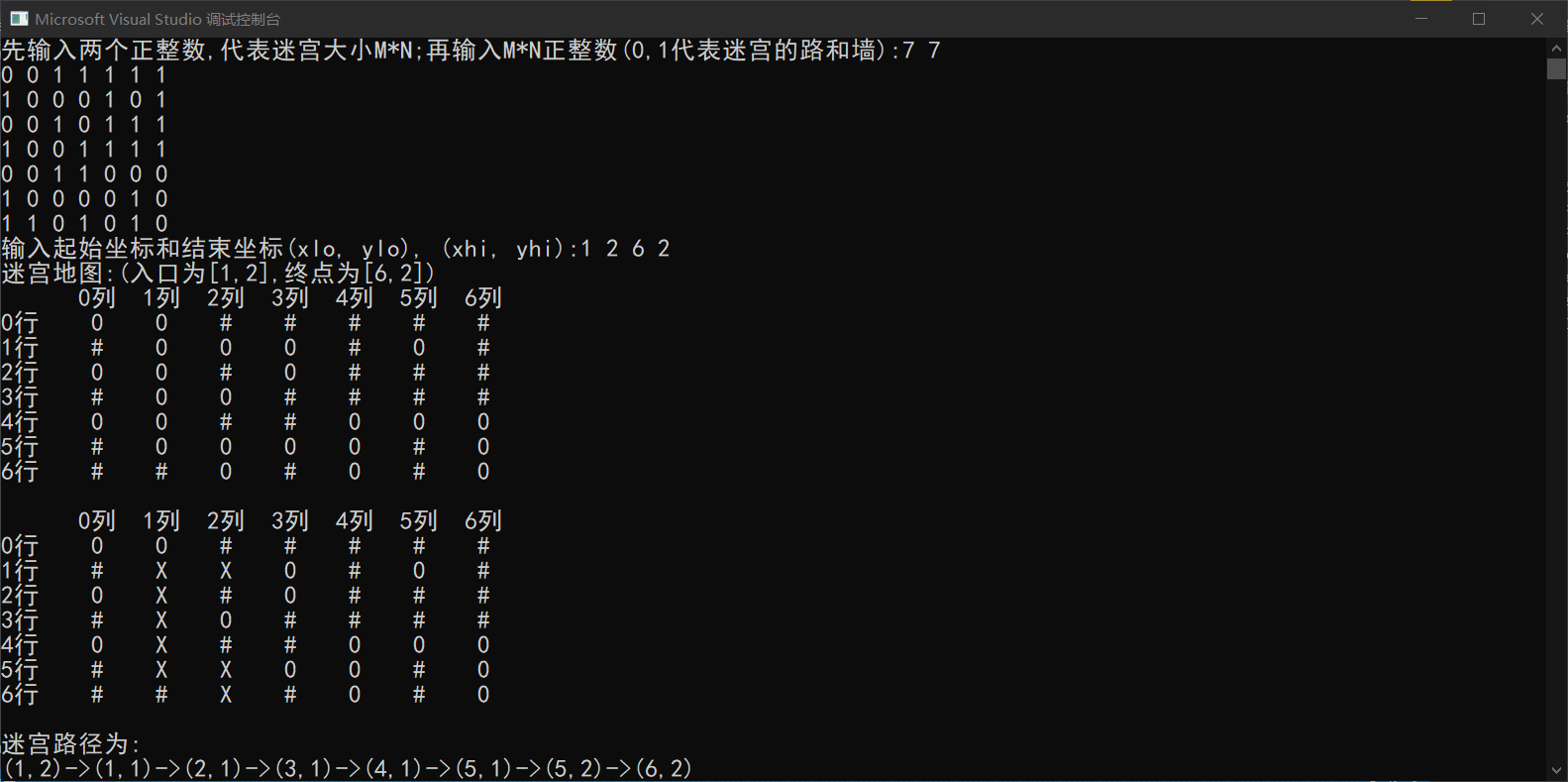
0 0 1 1 0 0 0

1 0 0 0 0 1 0

1 1 0 1 0 1 0

1 2 6 2

**实验结果：**



### 4.1.3 测试3

**测试用例：**

8 9

0 0 1 1 1 1 1 0 1

0 0 0 0 0 0 1 0 1

0 0 1 0 1 0 0 0 0

1 0 0 1 1 1 1 1 0

0 0 1 1 0 0 0 1 0

1 0 0 0 0 1 0 1 1

1 1 0 1 0 1 0 1 1

1 0 1 1 0 1 0 1 1

0 0 3 8

**实验结果：**

**(从测试用例可以看出来,虽然找到了路径,但是不是最短路径)**



## 4.2 边界测试

### 4.2.1 没有从入口到出口的路径

**测试用例：**

8 9

0 0 1 1 1 1 1 0 1

0 0 0 0 0 0 1 0 1

0 0 1 0 1 0 0 0 0

1 0 0 1 1 1 1 1 0

0 0 1 1 0 0 0 1 0

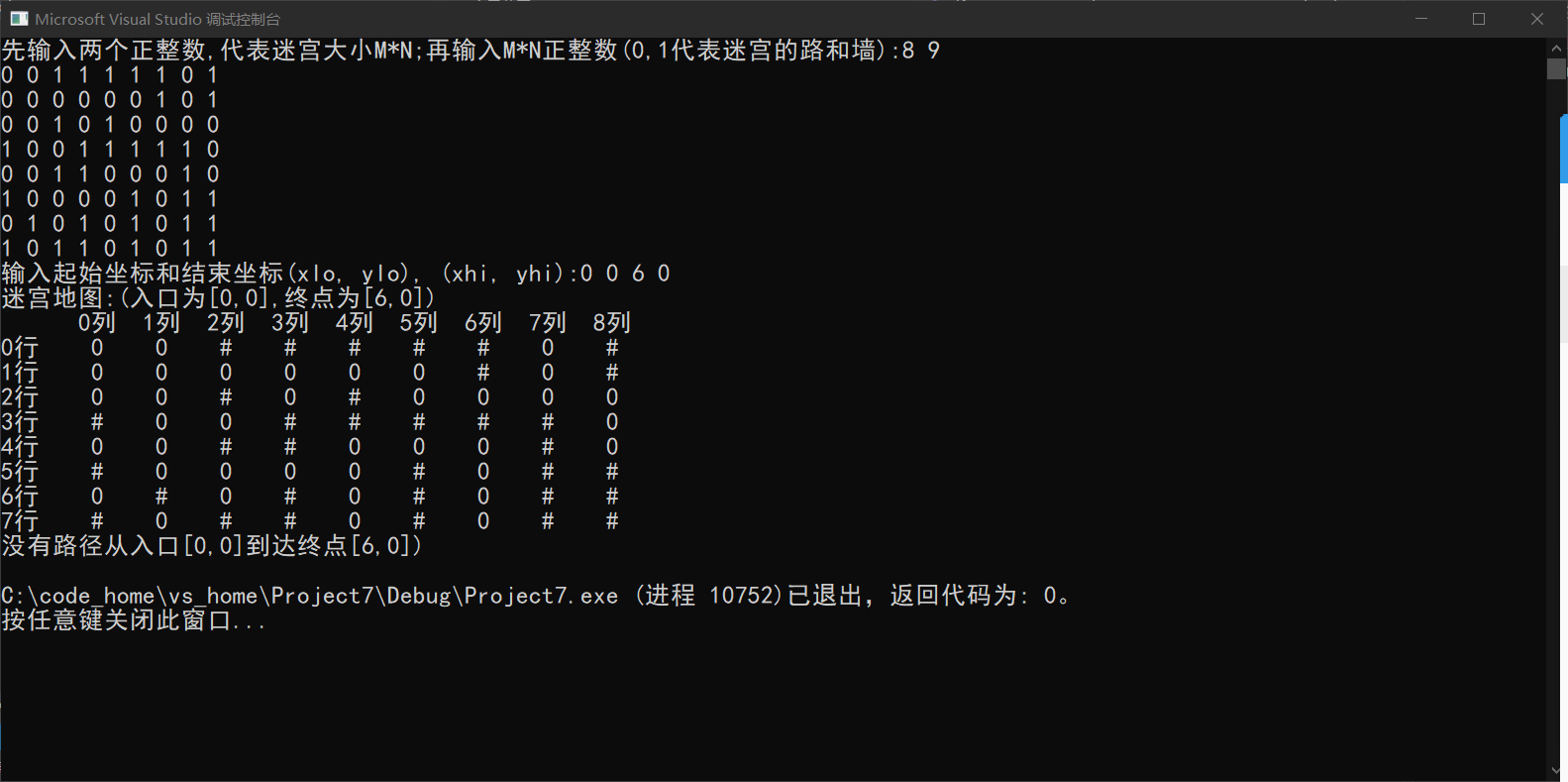
1 0 0 0 0 1 0 1 1

0 1 0 1 0 1 0 1 1

1 0 1 1 0 1 0 1 1

0 0 6 0

**实验结果：**



### 4.2.2 入口不在数组中或者为墙

**测试用例：**

8 9

0 0 1 1 1 1 1 0 1

0 0 0 0 0 0 1 0 1

0 0 1 0 1 0 0 0 0

1 0 0 1 1 1 1 1 0

0 0 1 1 0 0 0 1 0

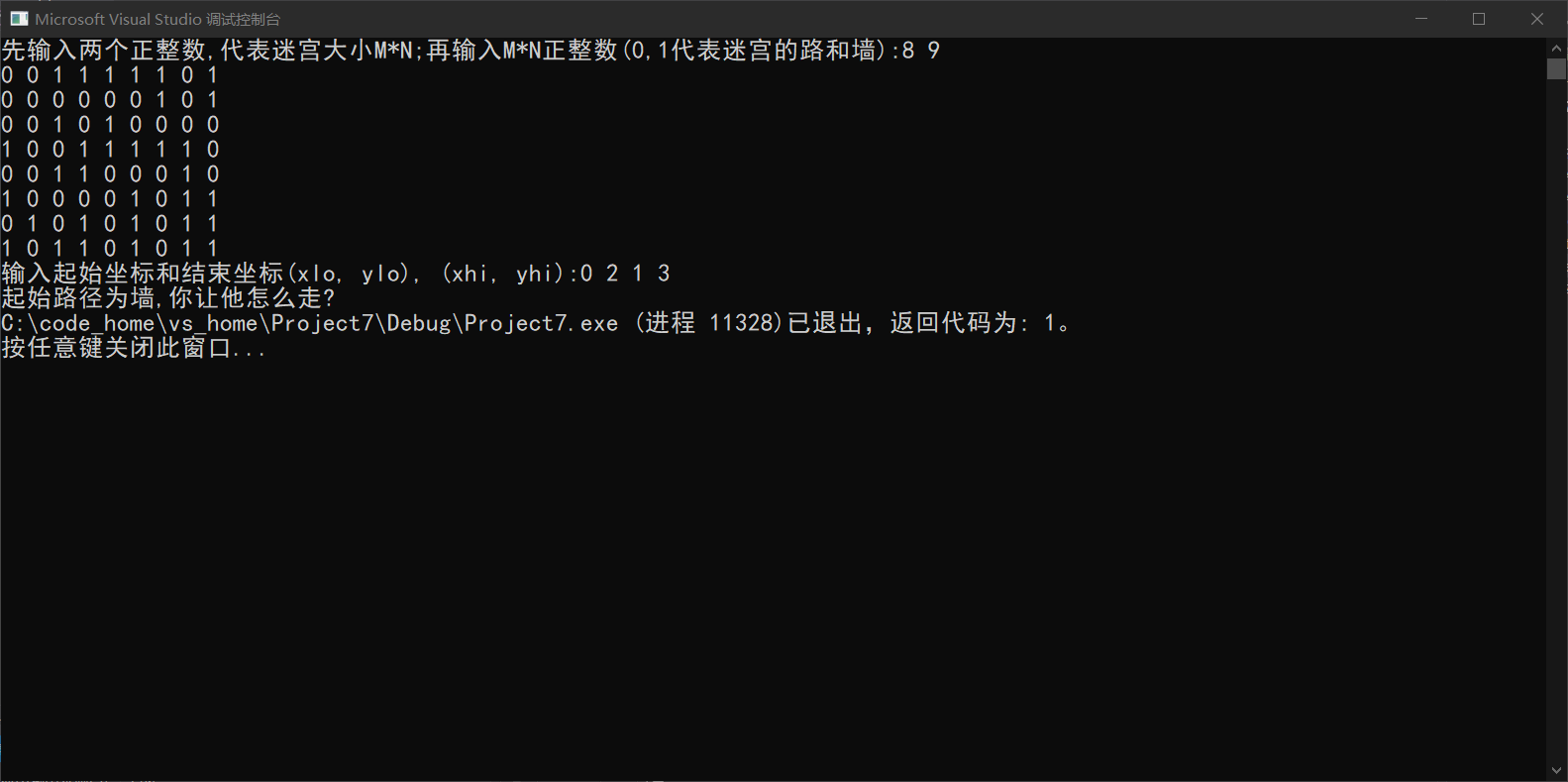
1 0 0 0 0 1 0 1 1

0 1 0 1 0 1 0 1 1

1 0 1 1 0 1 0 1 1

0 2 1 3

**实验结果：**



**测试用例：**

8 9

0 0 1 1 1 1 1 0 1

0 0 0 0 0 0 1 0 1

0 0 1 0 1 0 0 0 0

1 0 0 1 1 1 1 1 0

0 0 1 1 0 0 0 1 0

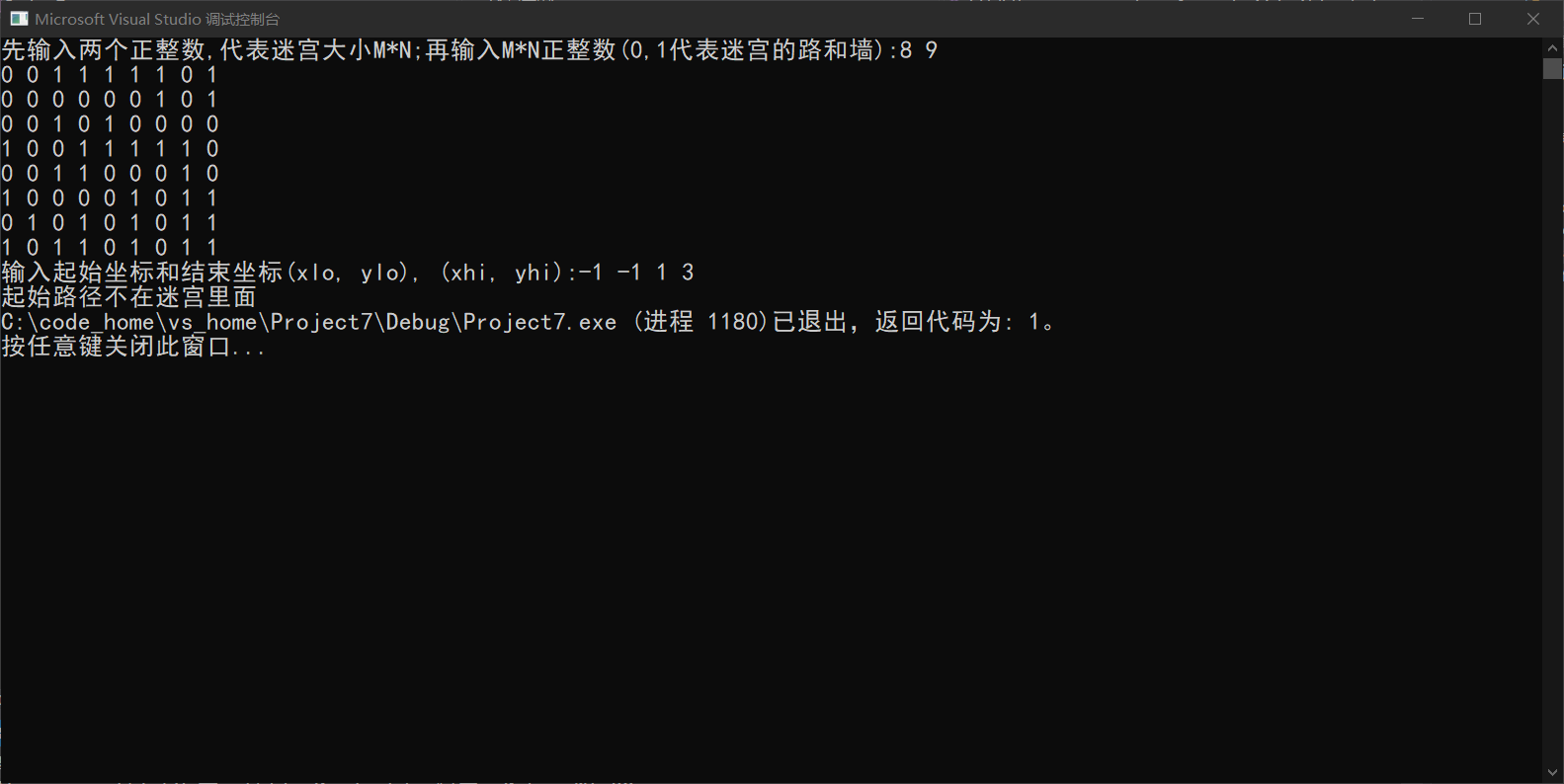
1 0 0 0 0 1 0 1 1

0 1 0 1 0 1 0 1 1

1 0 1 1 0 1 0 1 1

-1 -1 1 3

**实验结果：**



### 4.2.3 出口不在迷宫中或者出口为墙

**测试用例：**

8 9

0 0 1 1 1 1 1 0 1

0 0 0 0 0 0 1 0 1

0 0 1 0 1 0 0 0 0

1 0 0 1 1 1 1 1 0

0 0 1 1 0 0 0 1 0

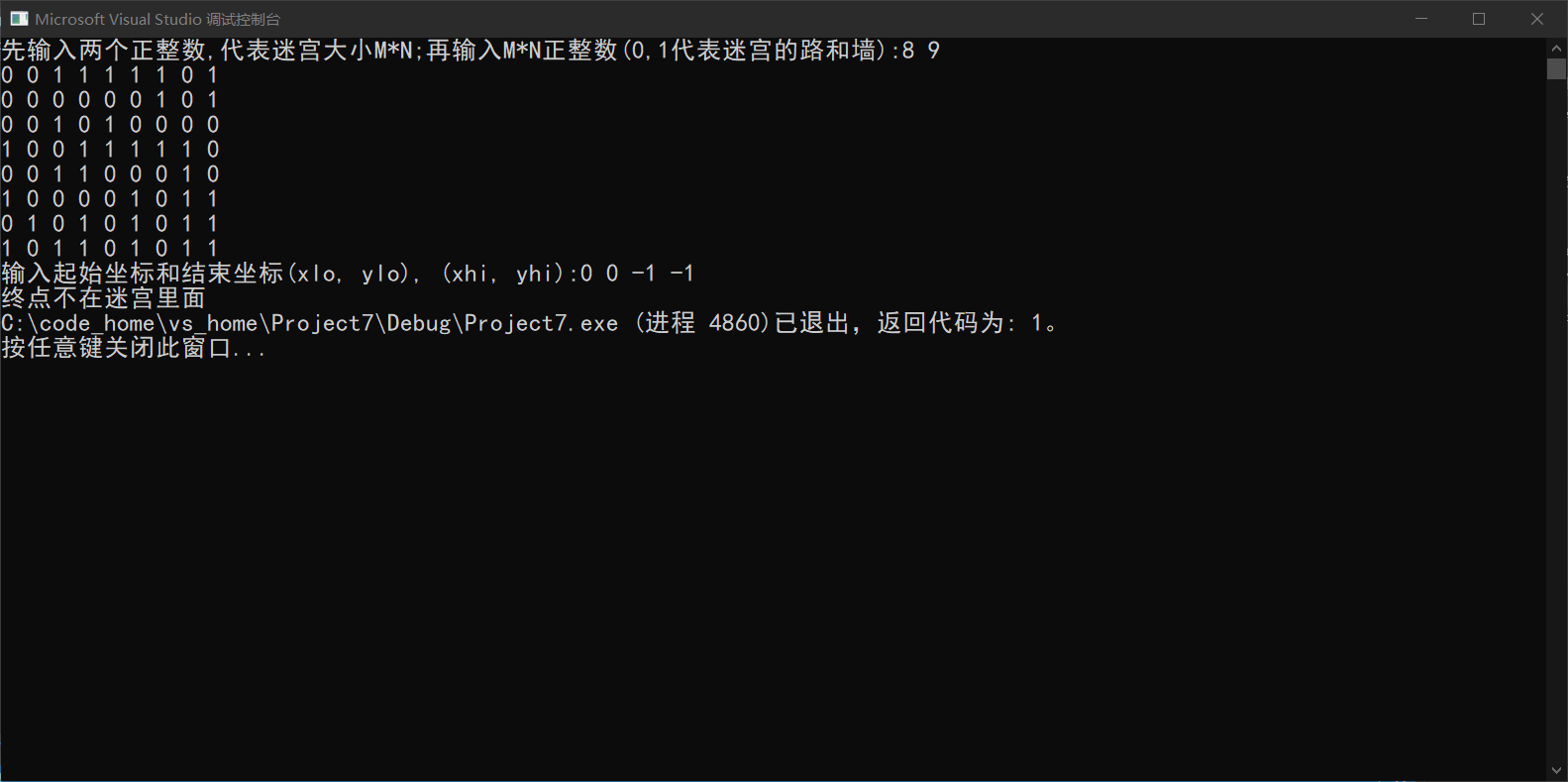
1 0 0 0 0 1 0 1 1

0 1 0 1 0 1 0 1 1

1 0 1 1 0 1 0 1 1

0 0 -1 -1

**实验结果：**



**测试用例：**

8 9

0 0 1 1 1 1 1 0 1

0 0 0 0 0 0 1 0 1

0 0 1 0 1 0 0 0 0

1 0 0 1 1 1 1 1 0

0 0 1 1 0 0 0 1 0

1 0 0 0 0 1 0 1 1

0 1 0 1 0 1 0 1 1

1 0 1 1 0 1 0 1 1

0 0 7 8

**实验结果：**

